Software Communications Architecture Specification

APPENDIX C CORE FRAMEWORK IDL

Revision Summary

0.1	original draft for industry review and comment
0.2	updated draft for industry review and comment.
0.3	updated draft for industry review and comment.
0.4	updated draft for industry review and comment. Updated CF IDL to reflect SCA changes and added Push Port and Pull Port modules for message interface extensions.
1.0	release for initial implementation and validation. Updated all IDL to reflect SCA changes and added PortTypes CORBA module of basic sequence types. [1.0a corrected PullPort to be a CORBA module as stated in text.]

Table of Contents

APPENDIX C CORE FRAMEWORK IDL

The CF interfaces are expressed in CORBA IDL. The IDL has been generated directly by the Rational Rose UML software modeling tool. This "forward engineering" approach ensures that the IDL accurately reflects the architecture definition as contained in the UML models. Any IDL compiler for the target language of choice may compile the generated IDL.

The CF interfaces are contained in the CF CORBA module. Additionally, IDL modules are provided for interfaces that extend the *CF::Port* interface by defining basic data sequence types and for pushing data to a consumer or pulling data from a producer. Figure C-1 shows the relationship between these CORBA modules.

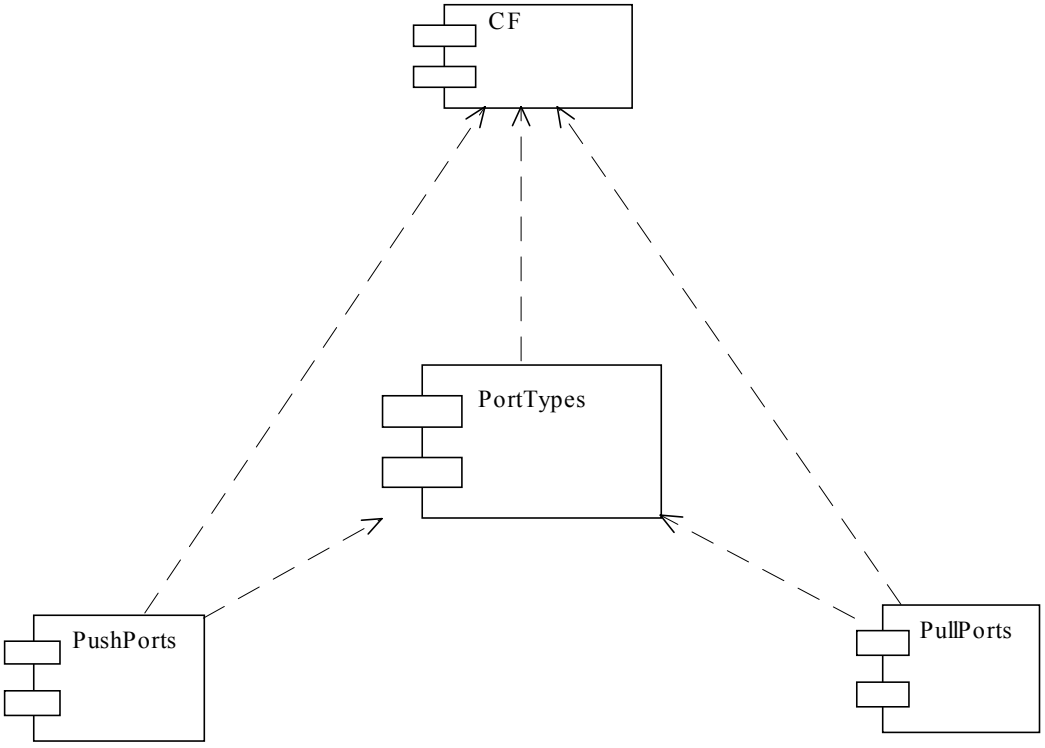

Figure C-1. Relationships Between CORBA Modules

The IDL modules are also available in electronic form.

C.1 CORE FRAMEWORK IDL.

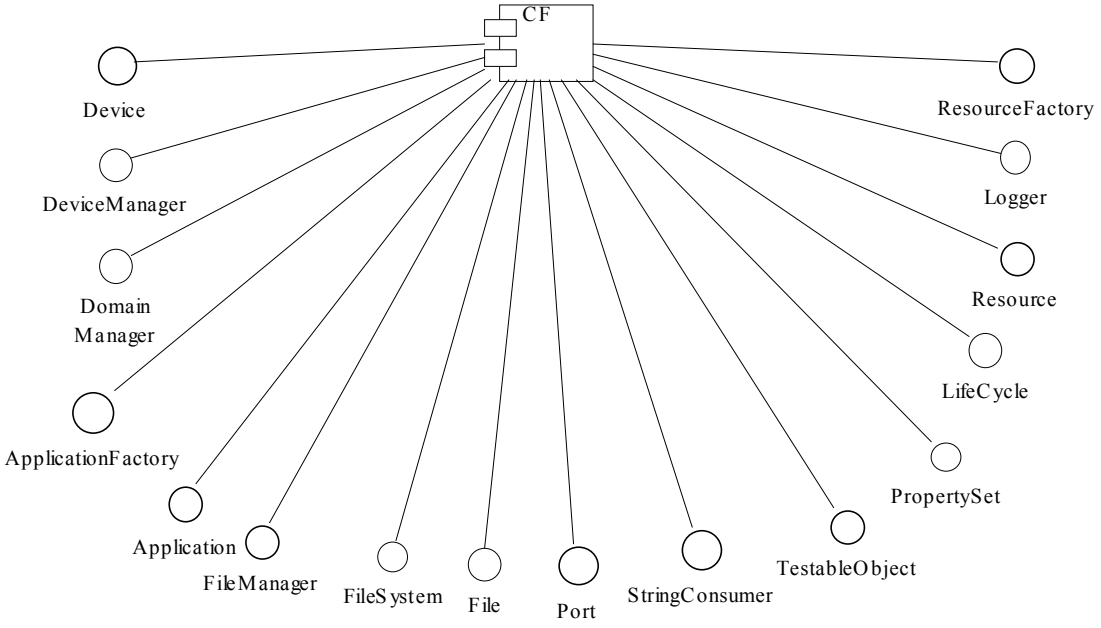

Figure C-2. CF CORBA Module

The following is the CF IDL generated from the Rational Rose model.

```
//## Module: CF
//## Subsystem:
Core_CSCI_IDL_Implementation_Components::CF_IDL_Implementation_Component
//## Source file:
/software/components/RadioCORBA/mmits_sdr/sdr_code.ss/glbick.wrk/CF.idl
//## Documentation::
//      This CORBA module defines the SDR CF interfaces and
//      types.
//##begin module.cm preserve=no
//    %X% %Q% %Z% %W%
//##end module.cm

//##begin module.cp preserve=no
//##end module.cp


#ifndef CF_idl
#define CF_idl

//##begin module.additionalIncludes preserve=no
//##end module.additionalIncludes

//##begin module.includes preserve=yes
//##end module.includes
```

```
// ============================================================

module CF
{
  interface Device;

  //## DeviceSequence Documentation:
  //      This type defines an unbounded sequence of Devices.

  //      The IDL to Ada mapping has a problem with self
  //      referential interfaces. To get around this problem,
  //      the interface Device forward declaration has been
  //      created and this type has been moved outside of the
  //      interface.
  //## Category: CF_IDL_Design_Components
  typedef sequence <Device> DeviceSequence;

  //##begin module.declarations preserve=no
  //##end module.declarations

  //##begin module.additionalDeclarations preserve=yes
  //##end module.additionalDeclarations


  //## DataType Documentation:
  //      This type is a CORBA  IDL struct type which can be
  //      used to hold any CORBA basic type or static IDL
  //      type.
  //## Category: CF_IDL_Design_Components

  struct DataType {
    //##begin DataType.initialDeclarations preserve=yes
    //##end DataType.initialDeclarations

    // Attributes

    //## Attribute: id
    //## Documentation:
    //      The id attribute indicates the kind of value (e.g.,
    //      frequency, preset, etc.).
    string id;
    //## Attribute: value
    //## Documentation:
    //      The value attribute can be any static IDL type or
    //      CORBA basic type.
    any value;

    // Relationships


    // Associations



    //##begin DataType.additionalDeclarations preserve=yes
    //##end DataType.additionalDeclarations
```

```
};

//## Port Documentation:
//      This interface is a generic Port that has behavior
//      for connecting or disconnecting ports together.
//      This interface can be extended for specific data
//      types with push or pull behavior.
//## Category: CF_IDL_Design_Components

interface Port {
  //##begin Port.initialDeclarations preserve=yes
  //##end Port.initialDeclarations

  // Nested Classes
  //## InvalidPort Documentation:
  //      This exception indicates that the port is invalid:
  //      error code 1 means unable to narrow the object
  //      reference, error code 2 means port is not connected
  //      to this port., or error code 3 means portID is not
  //      recognized by the port,
  //## Category: CF_IDL_Design_Components

  exception InvalidPort {
    //##begin InvalidPort.initialDeclarations preserve=yes
    //##end InvalidPort.initialDeclarations

    // Attributes

    unsigned short errorCode;
    string msg;

    // Relationships


    // Associations



    //##begin InvalidPort.additionalDeclarations preserve=yes
    //##end InvalidPort.additionalDeclarations

  };

  //## OccupiedPort Documentation:
  //      This exception indicates the Port is unable to
  //      accept the connection because it is already fully
  //      occupied.
  //## Category: CF_IDL_Design_Components

  exception OccupiedPort {
    //##begin OccupiedPort.initialDeclarations preserve=yes
    //##end OccupiedPort.initialDeclarations

    // Attributes
```

```
    // Relationships


    // Associations



    //##begin OccupiedPort.additionalDeclarations preserve=yes
    //##end OccupiedPort.additionalDeclarations

  };

  // Attributes


  // Relationships


  // Associations


  // Operations

  //## Operation: connectPort
  //## Documentation:
  //      This operation connects a port to another port.
  void connectPort(in Object connection, in string name)
      raises( InvalidPort, OccupiedPort );

  //## Operation: disconnectPort
  //## Documentation:
  //      This operation disconnects a port.  The port is no
  //      longer used and is released.
  void disconnectPort(in string name)
      raises( InvalidPort );



  //##begin Port.additionalDeclarations preserve=yes
  //##end Port.additionalDeclarations

};

//## InvalidFileName Documentation:
//      This exception indicates an invalid file name was
//      passed to a File Service operation. The message
//      provides information describing why the filename
//      was invalid.
//## Category: CF_IDL_Design_Components

exception InvalidFileName {
  //##begin InvalidFileName.initialDeclarations preserve=yes
  //##end InvalidFileName.initialDeclarations

  // Attributes

  string msg;
```

```
  // Relationships


  // Associations



  //##begin InvalidFileName.additionalDeclarations preserve=yes
  //##end InvalidFileName.additionalDeclarations

};

//## FileException Documentation:
//      This exception indicates a file-related error
//      occurred. The message shall provide information
//      describing the error.  The message can be used for
//      logging the error.
//## Category: CF_IDL_Design_Components

exception FileException {
  //##begin FileException.initialDeclarations preserve=yes
  //##end FileException.initialDeclarations

  // Attributes

  string msg;
  //## Attribute: errorCode
  //## Documentation:
  //      The error code that corresponds to the error
  //      message.
  unsigned short errorCode;

  // Relationships


  // Associations



  //##begin FileException.additionalDeclarations preserve=yes
  //##end FileException.additionalDeclarations

};

//## Category: CF_IDL_Design_Components

exception InvalidProfile {
  //##begin InvalidProfile.initialDeclarations preserve=yes
  //##end InvalidProfile.initialDeclarations

  // Attributes


  // Relationships
```

```
  // Associations



  //##begin InvalidProfile.additionalDeclarations preserve=yes
  //##end InvalidProfile.additionalDeclarations

};

//## TestableObject Documentation:
//      The TestableObject interface defines operations
//      that can be used to test object implementations.
//## Category: CF_IDL_Design_Components

interface TestableObject {
  //##begin TestableObject.initialDeclarations preserve=yes
  //##end TestableObject.initialDeclarations

  // Nested Classes
  //## UnknownTest Documentation:
  //      This exception indicates the test is unknow by the
  //      object.
  //## Category: CF_IDL_Design_Components

  exception UnknownTest {
    //##begin UnknownTest.initialDeclarations preserve=yes
    //##end UnknownTest.initialDeclarations

    // Attributes


    // Relationships


    // Associations



    //##begin UnknownTest.additionalDeclarations preserve=yes
    //##end UnknownTest.additionalDeclarations

  };

  // Attributes


  // Relationships


  // Associations


  // Operations

  //## Operation: runTest
  //## Documentation:
  //      The selfTest operation performs a specific test on
```

```
  //      an object.  True is returned if the test passes,
  //      otherwise false is returned.  When false is
  //      returned, the operation also returns a reason why
  //      the test failed.
  long runTest(in unsigned long testNum)
      raises( UnknownTest );



  //##begin TestableObject.additionalDeclarations preserve=yes
  //##end TestableObject.additionalDeclarations

};

//## Properties Documentation:
//      The Properties is a CORBA IDL unbounded sequence of
//      DataType(s), which can be used in defining a
//      sequence of name and value pairs.  The
//      relationships for Propertiies are shown in the
//      Properties Relationships figure.
//## Category: CF_IDL_Design_Components

typedef sequence <DataType> Properties;


//## PropertySet Documentation:
//      The PropertySet interface defines configure and
//      query operations to access component
//      properties/attributes.
//## Category: CF_IDL_Design_Components

interface PropertySet {
  //##begin PropertySet.initialDeclarations preserve=yes
  //##end PropertySet.initialDeclarations

  // Nested Classes
  //## InvalidConfiguration Documentation:
  //      This exception indicates the configuration of a
  //      component has failed (no configuration at all was
  //      done). The message provides additional information
  //      describing the reason why the error occurred. The
  //      invalid properties returned indicates the
  //      properties that were invalid.
  //## Category: CF_IDL_Design_Components

  exception InvalidConfiguration {
    //##begin InvalidConfiguration.initialDeclarations preserve=yes
    //##end InvalidConfiguration.initialDeclarations

    // Attributes

    string msg;
    Properties invalidProperties;

    // Relationships
```

```
  // Associations


  //##begin InvalidConfiguration.additionalDeclarations preserve=yes
  //##end InvalidConfiguration.additionalDeclarations

};

//## PartialConfiguration Documentation:
//      This exception indicates the configuration of a
//      component was partially successful. The invalid
//      properties returned indicates the properties that
//      were invalid.
//## Category: CF_IDL_Design_Components

exception PartialConfiguration {
  //##begin PartialConfiguration.initialDeclarations preserve=yes
  //##end PartialConfiguration.initialDeclarations

  // Attributes

  Properties invalidProperties;

  // Relationships


  // Associations



  //##begin PartialConfiguration.additionalDeclarations preserve=yes
  //##end PartialConfiguration.additionalDeclarations

};

//## UnknownProperties Documentation:
//      This exception indicates a set of properties
//      unknown by the component.
//## Category: CF_IDL_Design_Components

exception UnknownProperties {
  //##begin UnknownProperties.initialDeclarations preserve=yes
  //##end UnknownProperties.initialDeclarations

  // Attributes

  Properties invalidProperties;

  // Relationships


  // Associations



  //##begin UnknownProperties.additionalDeclarations preserve=yes
```

```
    //##end UnknownProperties.additionalDeclarations

  };

  // Attributes


  // Relationships


  // Associations


  // Operations

  //## Operation: configure
  //## Documentation:
  //      The configure operation sets the component's
  //      properties.   Any basic CORBA type or static IDL
  //      type could be used for the configuration data.  An
  //      component's ICD or profile indicates the valid
  //      configuration values.

  //      This operation raises InvalidConfiguration
  //      exception when a configuration error occurs
  //      preventing any property configuration on the
  //      object.

  //      This operation raises PartialConfiguration
  //      exception when some configuration properties were
  //      successful and some configuration properties were
  //      not successful.
  void configure(in Properties configProperties)
      raises( InvalidConfiguration, PartialConfiguration );

  //## Operation: query
  //## Documentation:
  //      The query operation retrieves component's
  //      properties.   Any basic CORBA type or static IDL
  //      type could be used for the query. An component's
  //      ICD or XML profile indicates the valid query types.

  //      A properties set of size 0 means all properties
  //      arte returned back.

  //      This operation raises the UnknownProperties
  //      exception when one or more a properties being
  //      requested are not known by the component.
  void query(inout Properties configProperties)
      raises( UnknownProperties );



  //##begin PropertySet.additionalDeclarations preserve=yes
  //##end PropertySet.additionalDeclarations

};
```

```
//## Category: CF_IDL_Design_Components

exception InvalidObjectReference {
  //##begin InvalidObjectReference.initialDeclarations preserve=yes
  //##end InvalidObjectReference.initialDeclarations

  // Attributes

  string msg;

  // Relationships


  // Associations



  //##begin InvalidObjectReference.additionalDeclarations preserve=yes
  //##end InvalidObjectReference.additionalDeclarations

};

//## Category: CF_IDL_Design_Components

interface StringConsumer {
  //##begin StringConsumer.initialDeclarations preserve=yes
  //##end StringConsumer.initialDeclarations

  // Attributes


  // Relationships


  // Associations


  // Operations

  oneway void processString(in string stringMsg, in Properties options);



  //##begin StringConsumer.additionalDeclarations preserve=yes
  //##end StringConsumer.additionalDeclarations

};

//## OctetSequence Documentation:
//      This type is a CORBA unbounded sequence of octets.
//## Category: CF_IDL_Design_Components

typedef sequence<octet> OctetSequence;


//## File Documentation:
```

```
//      The File interface provides the ability to read and
//      write files residing within a CF compliant
//      distributed FileSystem.  A file can be thought of
//      conceptually as a sequence of octets with a current
//      filepointer describing where the next read or write
//      will occur.  This filepointer points to the
//      beginning of the file upon construction of the file
//      object. The File interface is modeled after the
//      POSIX/C file interface.
//## Category: CF_IDL_Design_Components

interface File {
  //##begin File.initialDeclarations preserve=yes
  //##end File.initialDeclarations

  // Nested Classes
  //## IOException Documentation:
  //      This exception indicates an error occurred during a
  //      read or write operation to a File. The message
  //      provides information describing why the I/O error
  //      occurred.
  //## Category: CF_IDL_Design_Components

  exception IOException {
    //##begin IOException.initialDeclarations preserve=yes
    //##end IOException.initialDeclarations

    // Attributes

    string msg;
    //## Attribute: errorCode
    //## Documentation:
    //      The error code that corresponds to the error
    //      message.
    unsigned short errorCode;

    // Relationships


    // Associations



    //##begin IOException.additionalDeclarations preserve=yes
    //##end IOException.additionalDeclarations

  };

  //## InvalidFilePointer Documentation:
  //      This exception indicates the file pointer is out of
  //      range based upon the current file size.
  //## Category: CF_IDL_Design_Components

  exception InvalidFilePointer {
    //##begin InvalidFilePointer.initialDeclarations preserve=yes
    //##end InvalidFilePointer.initialDeclarations
```

```
  // Attributes


  // Relationships


  // Associations



  //##begin InvalidFilePointer.additionalDeclarations preserve=yes
  //##end InvalidFilePointer.additionalDeclarations

};

// Attributes

//## Attribute: fileName
//## Documentation:
//      Theis attribute provides read access to the fully
//      qualified name of the file.
readonly attribute string fileName;
//## Attribute: filePointer
//## Documentation:
//      The filePointer attribute provides read and write
//      access to the file pointer position where the next
//      read or write will occur.
readonly attribute unsigned long filePointer;

// Relationships


// Associations


// Operations

//## Operation: read
//## Documentation:
//      The read operation reads a maximum of length octets
//      from the file and moves the file pointer forward by
//      the number of octets actually read.   An I/O
//      exception is raised when a read error has occurred.
void read(out OctetSequence data, in unsigned long length)
    raises( IOException );

//## Operation: write
//## Documentation:
//      The write operation attempts to write the number of
//      octets based upon its length to file and moves the
//      file pointer forward by the number of octets
//      written. The write operation writes no data and
//      file pointer remains unchanged when an error
//      occurs, and IOException is raised.
void write(in OctetSequence data)
    raises( IOException );
```

```
  //## Operation: sizeOf
  //## Documentation:
  //      The sizeOf operation returns the current size of
  //      the file.
  unsigned long sizeOf()
      raises( FileException );

  //## Operation: close
  //## Documentation:
  //      The close operation closes the File and releases
  //      the File on the server side.  A closed file is
  //      longer capable of File related operations.   A
  //      client should release its CORBA File reference
  //      after closing the File.  The close operation raises
  //      CF::FileException exception when it cannot
  //      successfully close the file.
  void close()
      raises( FileException );

  //## Operation: setFilePointer
  //## Documentation:
  //      The setFilePointer operation shall set the file
  //      pointer to the input pointer parameter.
  void setFilePointer(in unsigned long filePointer)
      raises( InvalidFilePointer );



  //##begin File.additionalDeclarations preserve=yes
  //##end File.additionalDeclarations

};

//## StringSequence Documentation:
//      This type defines a sequence of strings
//## Category: CF_IDL_Design_Components

typedef sequence <string> StringSequence;


//## FileSystem Documentation:
//      The FileSystem interface defines the CORBA
//      operations to enable remote access to a physical
//      file system._
//## Category: CF_IDL_Design_Components

interface FileSystem {
  //##begin FileSystem.initialDeclarations preserve=yes
  //##end FileSystem.initialDeclarations

  // Nested Classes
  //## UnknownFileSystemProperties Documentation:
  //      This exception indicates a set of properties
  //      unknown by the FileSystem object.
  //## Category: CF_IDL_Design_Components

  exception UnknownFileSystemProperties {
```

```
  //##begin UnknownFileSystemProperties.initialDeclarations preserve=yes
  //##end UnknownFileSystemProperties.initialDeclarations

  // Attributes

  Properties invalidProperties;

  // Relationships


  // Associations



  //##begin UnknownFileSystemProperties.additionalDeclarations preserve=yes
  //##end UnknownFileSystemProperties.additionalDeclarations

};

// Attributes


// Relationships


// Associations


// Operations

//## Operation: remove
//## Documentation:
//      The remove operation removes the file with the
//      given filename.  This operation ensures that the
//      filename is an absolute pathname of the file
//      relative to the target FileSystem. If an error
//      occurs, this operation raises the appropriate
//      exception:

//      InvalidFilename - The filename is not valid.
//      FileException - A file-related error occurred
//      during the operation.
void remove(in string fileName)
    raises( FileException, InvalidFileName );

//## Operation: copy
//## Documentation:
//      The copy operation copies the source file with the
//      specified name to the destination FileSystem.
//      This operation ensures that the sourceFileName and
//      destinationFileName are absolute pathnames relative
//      to the target FileSystem. If an error occurs, this
//      operation raises the appropriate exception:

//      InvalidFilename - The filename is not valid.
//      FileException - A file-related error occurred
//      during the operation.
```

```
void copy(in string sourceFileName, in string destinationFileName)
    raises( InvalidFileName, FileException );

//## Operation: exists
//## Documentation:
//      The exists operation checks to see if a file exists
//      based on the filename parameter. This operation
//      ensures that the filename is a full pathname of the
//      file relative to the target FileSystem and raise an
//      InvalidFileName exception if the name is invalid.

//      This operation shall return True if the file
//      exists, otherwise False shall be returned.
boolean exists(in string fileName)
    raises( InvalidFileName );

//## Operation: list
//## Documentation:
//      The list operation returns a list of filenames
//      based upon the search pattern given.  The following
//      wildcard characters are supported:

//      *     shall be used to match any sequence of characters
//      (including null).
//      ?     shall be used to match any single character.

//      These wildcards may only be applied to the base
//      filename in the search pattern given.  For example,
//      the following are valid search patterns:
//      /tmp/files/*.*    Returns all files within the
//      /tmp/files directory.
//      /tmp/files/foo*   Returns all files beginning with
//      the letters "foo" in the /tmp/files directory.
//      /tmp/files/f??    Returns all 3 letter files beginning
//      with the letter f in the /tmp/files directory.

//      The following search pattern would be considered
//      illegal since it attempts to specify a wildcard on
//      something other that the base filename:
//      The list operation shall return a list of filenames
//      based upon the search pattern given.  The following
//      wildcard characters shall be supported:

//      *     shall be used to match any sequence of characters
//      (including null).
//      ?     shall be used to match any single character.

//      These wildcards may only be applied to the base
//      filename in the search pattern given.  For example,
//      the following are valid search patterns:
//      /tmp/files/*.*    Returns all files within the
//      /tmp/files directory.
//      /tmp/files/foo*   Returns all files beginning with
//      the letters "foo" in the /tmp/files directory.
//      /tmp/files/f??    Returns all 3 letter files beginning
//      with the letter f in the /tmp/files directory.
```

```
//      The following search pattern would be considered
//      illegal since it attempts to specify a wildcard on
//      something other that the base filename:
//      /*/files/foo*
StringSequence list(in string pattern);

//## Operation: create
//## Documentation:
//      The create operation creates a new File based upon
//      the provided file name and returns a File to the
//      opened file.  A null file is returned and a related
//      exception shall be raised if an error occurs.

//      InvalidFilename - The filename is not valid.
//      FileException   - File already exists or another
//      file error occurred.
File create(in string fileName)
    raises( InvalidFileName, FileException );

//## Operation: open
//## Documentation:
//      The open operation opens a file based upon the
//      input fileName. The readOnly parameter indicates if
//      the file should be opened for read access only.
//      When readOnly is false the file is opened for write
//      access.

//      A File shall be returned on successful completion
//      of this operation, otherwise a null File reference
//      shall be returned and a related exception shall be
//      raised. If the file is opened with the readOnly
//      flag set to true, then writes to the file will be
//      considered an error.

//      Exceptions/Errors
//      InvalidFilename - The filename is not valid.
//      FileException - File does not exist or another file
//      error occurred.
File open(in string fileName, in boolean read_Only)
    raises( InvalidFileName, FileException );

//## Operation: mkdir
//## Documentation:
//      The mkdir operation create a FileSystem directory
//      based on the directoryName given.  This operation
//      creates all parent directories required to create
//      the directory path given. If an error occurs, this
//      operation raises the appropriate exception.

//      Exceptions/Errors
//      InvalidFilename - The directory name is not valid.
//      FileException - A file-related error occurred
//      during the operation.
void mkdir(in string directoryName)
    raises( InvalidFileName, FileException );

//## Operation: rmdir
```

```
  //## Documentation:
  //      The rmdir operation removes a FileSystem directory
  //      based on the directoryName given. If an error
  //      occurs, this operation raises the appropriate
  //      exception.

  //      Exceptions/Errors
  //      InvalidFilename - The directory name is not valid.
  //      FileException - Directory does not exist or another
  //      file-related error occurred.
  void rmdir(in string directoryName)
      raises( InvalidFileName, FileException );

  //## Operation: query
  //## Documentation:
  //      The query operation retrieves file system
  //      properties.   Any basic CORBA type or static IDL
  //      type could be used for the query.  Typical file
  //      SystemProperties are:

  //      SIZE - the actual size of the file system
  //      AVAILABLE_SPACE - the available space on the file
  //      system

  //      CF::InvalidProperty exception is raised for
  //      properties not supported by the file system.
  void query(inout Properties fileSystemProperties)
      raises( UnknownFileSystemProperties );



  //##begin FileSystem.additionalDeclarations preserve=yes
  //##end FileSystem.additionalDeclarations

};

//## Logger Documentation:
//      The Logger interface is used to capture alarms, log
//      warnings and information messages during the
//      execution of software within the radio, and pushes
//      messages to registered consumers. The interface
//      provides operations for both producer and consumer
//      clients.
//## Category: CF_IDL_Design_Components

interface Logger {
  //##begin Logger.initialDeclarations preserve=yes
  //##end Logger.initialDeclarations

  // Nested Classes
  //## ProducerLogLevelType Documentation:
  //      This type defines the log levels for a producer.
  //## Category: CF_IDL_Design_Components

  struct ProducerLogLevelType {
    //##begin ProducerLogLevelType.initialDeclarations preserve=yes
    //##end ProducerLogLevelType.initialDeclarations
```

```
  // Attributes

  string name;
  unsigned short logLevel;

  // Relationships


  // Associations



  //##begin ProducerLogLevelType.additionalDeclarations preserve=yes
  //##end ProducerLogLevelType.additionalDeclarations

};

//## ProducerLogLevels Documentation:
//      This type defines an unbounded sequence of producer
//      log levels.
//## Category: CF_IDL_Design_Components

typedef sequence <ProducerLogLevelType> ProducerLogLevels;


//## MATCH_ALL_NAMES Documentation:
//      This constant defines a special name that indicates
//      all producer names are to be used by a consumer or
//      for getting log levels.
//## Category: CF_IDL_Design_Components

const string MATCH_ALL_NAMES = "*";


//## NameNotFound Documentation:
//      This exception indicates the input name (producer
//      or consumer) does not exist in the logger.
//## Category: CF_IDL_Design_Components

exception NameNotFound {
  //##begin NameNotFound.initialDeclarations preserve=yes
  //##end NameNotFound.initialDeclarations

  // Attributes


  // Relationships


  // Associations



  //##begin NameNotFound.additionalDeclarations preserve=yes
  //##end NameNotFound.additionalDeclarations
```

```
    };

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: logData
    //## Documentation:
    //      This operation logs a log string and a time stamp
    //      to the console depending on the current log level
    //      set for the producer object and the log level of
    //      the string.  It also logs the same information to a
    //      file if file logging is enabled for the object. The
    //      operation also pushes the data to registered
    //      consumers based upon their log levels. The logger
    //      log level  is automatically assigned to a new
    //      producer.
    oneway void logData(in string producerName, in string messageString, in
unsigned short logLevel);

    //## Operation: setLoggingState
    //## Documentation:
    //      This operation enables the logging of all messages
    //      at the currently set level for each object, or
    //      disables the logging of all messages from all
    //      objects, depending on the value of the argument.
    void setLoggingState(in boolean enable);

    //## Operation: setConsumerLogLevel
    //## Documentation:
    //      This operation sets the log level for a consumer
    //      object.  All incoming log strings <= to the
    //      currently set level are sent to the consumer. The
    //      log level is bitmapped 00 00 - 7F FF (hex) with bit
    //      16 being a control bit to allow for log level
    //      manipulation.

    //      Examples:
    //      LogLevel = C010 h (1100 0000 0001 0000 b) indicates
    //      only levels 14 and 4 are to be displayed.
    //      LogLevel = 000A h indicates levels 10 and below
    //      will be displayed, and bits 4-14 are unused.
    void setConsumerLogLevel(in string consumerName, in string producerName, in
unsigned short logLevel)
        raises( NameNotFound );

    //## Operation: getLogs
    //## Documentation:
    //      This operation returns the last number of log
    //      messages stored locally within the logger.
```

```
    StringSequence getLogs(in unsigned short number);

    //## Operation: registerConsumer
    //## Documentation:
    //      This operation registers a consumer object with the
    //      logger.   Initially all producers' messages that
    //      pass the input  logLevel are pushed to the
    //      consumer.  A consumer can change its filtering by
    //      the setConsumerLogLevel operation.
    void registerConsumer(in string consumerName, in StringConsumer
strConsumer, in unsigned short logLevel);

    //## Operation: unregisterConsumer
    //## Documentation:
    //      This operation unregisters a consumer object.
    void unregisterConsumer(in string consumerName)
        raises( NameNotFound );

    //## Operation: getConsumerLogLevels
    //## Documentation:
    //      This operation returns the current log levels for
    //      passed in registered consumer object name(s).  A
    //      special ConsumberName of "*" shall return all
    //      consumer loglevels for the passed in wildcard
    //      ("*")  consumer parameter (see examples).

    //      I.E.: "*GUI" would be all registered consumer names
    //      ending with  string "GUI".

    //      I.E.2: "GUI*" would be all registered consumer
    //      names starting with string "GUI".

    //      I.E.3: "*" would be all registered consumers.
    ProducerLogLevels getConsumerLogLevels(in string consumerName)
        raises( NameNotFound );

    void setProducerLogLevel(in string producerName, in unsigned short
logLevel)
        raises( NameNotFound );

    void registerProducer(in string producerName, in unsigned short logLevel);

    void unregisterProducer(in string producerName)
        raises( NameNotFound );

    ProducerLogLevels getProducerLogLevels(in string producerName)
        raises( NameNotFound );



    //##begin Logger.additionalDeclarations preserve=yes
    //##end Logger.additionalDeclarations

  };

  //## FileManager Documentation:
  //      Multiple, distributed FileSystems may be accessed
```

```
//      through a FileManager. The FileManager interface
//      appears to be a single FileSystem although the
//      actual file storage may span multiple physical file
//      systems.  This is called a federated file system. A
//      federated file system is created using the mount
//      and unmount operations. Typically, the Domain
//      Manager or system initialization software will
//      invoke these operations.

//      The FileManager inherits the IDL interface of a File
//      System. Based upon the pathname of a directory or
//      file and the set of mounted filesystems, the File
//      Manager will delegate the FileSystem operations to
//      the appropriate FileSystem.  For example, if a File
//      System is mounted at /ppc2, an open operation for a
//      file called /ppc2/profile.xml would be delegated to
//      the mounted FileSystem. The mounted FileSystem will
//      be given the filename relative to it. In this
//      example the FileSystem's open operation would
//      receive /profile.xml as the fileName argument.

//      Another example of this concept can be shown using
//      the copy operation.  When a client invokes the copy
//      operation, the FileManager will delegate operations
//      to the appropriate FileSystems (based upon supplied
//      pathnames) thereby allowing copy of files between
//      filesystems.

//      If a client does not need to mount and unmount File
//      Systems, it can treat the FileManager as a File
//      System by CORBA widening a FileManager reference to
//      a FileSystem reference.  One can always widen a File
//      Manager to a FileSystem since the FileManager is
//      derived from a FileSystem.
//## Category: CF_IDL_Design_Components

interface FileManager : CF::FileSystem {
  //##begin FileManager.initialDeclarations preserve=yes
  //##end FileManager.initialDeclarations

  // Nested Classes
  //## MountType Documentation:
  //      The Mount structure identifies a FileSystem mounted
  //      within a FileManager.__
  //## Category: CF_IDL_Design_Components

  struct MountType {
    //##begin MountType.initialDeclarations preserve=yes
    //##end MountType.initialDeclarations

    // Attributes

    string mountPoint;
    FileSystem fs;

    // Relationships
```

```
  // Associations



  //##begin MountType.additionalDeclarations preserve=yes
  //##end MountType.additionalDeclarations

};

//## MountSequence Documentation:
//      This type defines an unbounded sequence of mounted
//      FileSystems.
//## Category: CF_IDL_Design_Components

typedef sequence <MountType> MountSequence;


//## NonExistentMount Documentation:
//      This exception indicates a mount point does not
//      exist within the FileManager
//## Category: CF_IDL_Design_Components

exception NonExistentMount {
  //##begin NonExistentMount.initialDeclarations preserve=yes
  //##end NonExistentMount.initialDeclarations

  // Attributes


  // Relationships


  // Associations



  //##begin NonExistentMount.additionalDeclarations preserve=yes
  //##end NonExistentMount.additionalDeclarations

};

//## FileSystemPropertyType Documentation:
//      This type shall be used to associate a property
//      with a specific FileSystem.
//## Category: CF_IDL_Design_Components

struct FileSystemPropertyType {
  //##begin FileSystemPropertyType.initialDeclarations preserve=yes
  //##end FileSystemPropertyType.initialDeclarations

  // Attributes

  string fileSystemName;
  DataType property;

  // Relationships
```

```
  // Associations



  //##begin FileSystemPropertyType.additionalDeclarations preserve=yes
  //##end FileSystemPropertyType.additionalDeclarations

};

//## FileSystemPropertySequence Documentation:
//      The FileSystemPropertySequence is an unbounded
//      sequence of FileSystemPropertyTypes.
//## Category: CF_IDL_Design_Components

typedef sequence <FileSystemPropertyType> FileSystemPropertySequence;


//## InvalidFileSystem Documentation:
//      This exception indicates the FileSystem is a null
//      (nil) object reference.
//## Category: CF_IDL_Design_Components

exception InvalidFileSystem {
  //##begin InvalidFileSystem.initialDeclarations preserve=yes
  //##end InvalidFileSystem.initialDeclarations

  // Attributes


  // Relationships


  // Associations



  //##begin InvalidFileSystem.additionalDeclarations preserve=yes
  //##end InvalidFileSystem.additionalDeclarations

};

//## MountPointAlreadyExists Documentation:
//      This exception indicates the mount point is already
//      in use in the file manager.
//## Category: CF_IDL_Design_Components

exception MountPointAlreadyExists {
  //##begin MountPointAlreadyExists.initialDeclarations preserve=yes
  //##end MountPointAlreadyExists.initialDeclarations

  // Attributes


  // Relationships
```

```
    // Associations



    //##begin MountPointAlreadyExists.additionalDeclarations preserve=yes
    //##end MountPointAlreadyExists.additionalDeclarations

  };

  // Attributes


  // Relationships


  // Associations


  // Operations

  //## Operation: mount
  //## Documentation:
  //      The mount operation associates the specified File
  //      System with the given mountPoint and adds the entry
  //      to the FileManager's mounted FileSystem list. This
  //      operation ensures that the mountPoint is a valid
  //      subdirectory path within the target FileSystem.
  //      The mount operation raises CF::InvalidFilename and
  //      the entry is not added to the FileManger's mounted
  //      list when the mount point (directory) name is not
  //      valid.
  void mount(in string mountPoint, in FileSystem file_System)
      raises( InvalidFileName, InvalidFileSystem, MountPointAlreadyExists );

  //## Operation: unmount
  //## Documentation:
  //      The unmount operation removes a mounted FileSystem
  //      from the FileManager whose mounted name matches the
  //      input mountPoint name.  The unmount operation
  //      raises NonExistentMount when the mount point does
  //      not exist within the FileManager.
  void unmount(in string mountPoint)
      raises( NonExistentMount );

  //## Operation: getMounts
  //## Documentation:
  //      The getMounts operation returns the FileManager's
  //      mounted FileSystems._
  MountSequence getMounts();



  //##begin FileManager.additionalDeclarations preserve=yes
  //##end FileManager.additionalDeclarations

};
```

```
//## LifeCycle Documentation:
//      The LifeCycle interface defines the generic
//      operations for initializing or releasing an
//      instantiated component specific data and/or
//      processing elements.
//## Category: CF_IDL_Design_Components

interface LifeCycle {
  //##begin LifeCycle.initialDeclarations preserve=yes
  //##end LifeCycle.initialDeclarations

  // Nested Classes
  //## InitializeError Documentation:
  //      This exception indicates an error occurred during
  //      component initialization. The messages provides
  //      additional information describing the reason why
  //      the error occurred.
  //## Category: CF_IDL_Design_Components

  exception InitializeError {
    //##begin InitializeError.initialDeclarations preserve=yes
    //##end InitializeError.initialDeclarations

    // Attributes

    StringSequence errorMessages;

    // Relationships


    // Associations



    //##begin InitializeError.additionalDeclarations preserve=yes
    //##end InitializeError.additionalDeclarations

  };

  //## ReleaseError Documentation:
  //      This exception indicates an error occurred during
  //      component release. The messages provides additional
  //      information describing the reason why the errors
  //      occurred.
  //## Category: CF_IDL_Design_Components

  exception ReleaseError {
    //##begin ReleaseError.initialDeclarations preserve=yes
    //##end ReleaseError.initialDeclarations

    // Attributes

    StringSequence errorMessages;

    // Relationships
```

```
    // Associations



    //##begin ReleaseError.additionalDeclarations preserve=yes
    //##end ReleaseError.additionalDeclarations

  };

  // Attributes


  // Relationships


  // Associations


  // Operations

  //## Operation: initialize
  //## Documentation:
  //      The purpose of the initialize operation is to
  //      provide a mechanism to set an object to an known
  //      initial state.  (For example, data structures may
  //      be set to initial values, memory may be allocated,
  //      hardware components may be configured to some
  //      state, etc.).

  //      This operation raises the LifeCycleException when
  //      an initialization error occurs.
  void initialize()
      raises( InitializeError );

  //## Operation: releaseObject
  //## Documentation:
  //      The purpose of the releaseObject operation is to
  //      provide a means by which an instantiated component
  //      may be destroyed. The releaseObject operation
  //      releases itself from the CORBA ORB.

  //      This operation raises a LifeCycleException when a
  //      release error occurs.
  void releaseObject()
      raises( ReleaseError );



  //##begin LifeCycle.additionalDeclarations preserve=yes
  //##end LifeCycle.additionalDeclarations

};

//## Resource Documentation:
//      The Resource interface defines the minimal
//      interface for any software resource created up by a
```

```
//      Domain Manager.
//## Category: CF_IDL_Design_Components

interface Resource : CF::LifeCycle, CF::TestableObject, CF::PropertySet {
  //##begin Resource.initialDeclarations preserve=yes
  //##end Resource.initialDeclarations

  // Nested Classes
  //## UnknownPort Documentation:
  //      This exception is raised if an undefined port is
  //      requested.
  //## Category: CF_IDL_Design_Components

  exception UnknownPort {
    //##begin UnknownPort.initialDeclarations preserve=yes
    //##end UnknownPort.initialDeclarations

    // Attributes


    // Relationships


    // Associations



    //##begin UnknownPort.additionalDeclarations preserve=yes
    //##end UnknownPort.additionalDeclarations

  };

  //## StartError Documentation:
  //      This exception indicates a Start error has occurred
  //      for the Resource.  An error message is given
  //      explainng the start error.
  //## Category: CF_IDL_Design_Components

  exception StartError {
    //##begin StartError.initialDeclarations preserve=yes
    //##end StartError.initialDeclarations

    // Attributes

    string msg;

    // Relationships


    // Associations



    //##begin StartError.additionalDeclarations preserve=yes
    //##end StartError.additionalDeclarations

  };
```

```
//## StopError Documentation:
//      This exception indicates a Stop error has occurred
//      for the Resource.  An error message is given
//      explainng the stop error.
//## Category: CF_IDL_Design_Components

exception StopError {
  //##begin StopError.initialDeclarations preserve=yes
  //##end StopError.initialDeclarations

  // Attributes

  string msg;

  // Relationships


  // Associations



  //##begin StopError.additionalDeclarations preserve=yes
  //##end StopError.additionalDeclarations

};

// Attributes


// Relationships


// Associations


// Operations

//## Operation: start
//## Documentation:
//      The purpose of this operation is to allow an object
//      implementing this interface to transition to steady
//      state operation.  This operation may include, but
//      is not limited to transformation of data received
//      from upstream producers and/or the production of
//      data for downstream consumers.
void start()
    raises( StartError );

//## Operation: stop
//## Documentation:
//      The purpose of this operation is to allow an object
//      implementing this interface to transition from
//      steady state operation to idle.
void stop()
    raises( StopError );
```

```
  //## Operation: getPort
  //## Documentation:
  //      The getInputPort operation returns the named input
  //      port if it exists.  A resource may have multiple
  //      input ports. The multiple ports provide flexibility
  //      for resources that must manage varying priority
  //      levels of incoming messages, provide multithreaded
  //      message handling, or other special message
  //      processing.  The getInputPort operation raises an
  //      UnknownPort exception if the port name is not
  //      recognized.
  Object getPort(in string name)
      raises( UnknownPort );



  //##begin Resource.additionalDeclarations preserve=yes
  //##end Resource.additionalDeclarations

};

//## ResourceFactory Documentation:
//      A Factory is used to create or destroy a Resource.
//      The Factory interface is designed after the Factory
//      Design Patterns.  The factory mechanism provides
//      client-server isolation among Resources (e.g.,
//      Network, Link, Modem, Access, etc.) and provides an
//      industry standard mechanism of obtaining a Resource
//      without knowing its identity.
//## Category: CF_IDL_Design_Components

interface ResourceFactory {
  //##begin ResourceFactory.initialDeclarations preserve=yes
  //##end ResourceFactory.initialDeclarations

  // Nested Classes
  //## Category: CF_IDL_Design_Components

  typedef unsigned short ResourceNumType;


  //## InvalidResourceNumber Documentation:
  //      This exception indicates the resource number does
  //      not exist in the ResourceFactory.
  //## Category: CF_IDL_Design_Components

  exception InvalidResourceNumber {
    //##begin InvalidResourceNumber.initialDeclarations preserve=yes
    //##end InvalidResourceNumber.initialDeclarations

    // Attributes


    // Relationships


    // Associations
```

```
      //##begin InvalidResourceNumber.additionalDeclarations preserve=yes
      //##end InvalidResourceNumber.additionalDeclarations

    };

    //## ShutdownFailure Documentation:
    //      This exception indicates the ResourceFactory was
    //      unable to be destroyed due to the fact the factory
    //      still contains resources.
    //## Category: CF_IDL_Design_Components

    exception ShutdownFailure {
      //##begin ShutdownFailure.initialDeclarations preserve=yes
      //##end ShutdownFailure.initialDeclarations

      // Attributes


      // Relationships


      // Associations



      //##begin ShutdownFailure.additionalDeclarations preserve=yes
      //##end ShutdownFailure.additionalDeclarations

    };

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: createResource
    //## Documentation:
    //      This operation returns a resource based upon the
    //      input resource number and qualifiers.  If the
    //      resource does not already exist then this operation
    //      creates the resource, else the operation returns
    //      the object already created for that resource number.
    Resource createResource(in ResourceNumType resourceNumber, in Properties
qualifiers);

    //## Operation: releaseResource
    //## Documentation:
    //      This operation removes the resource from the
```

```
  //      Factory if no other clients are using the
  //      resource.  The resource to be released is
  //      associated with a specific resource number.
  void releaseResource(in ResourceNumType resourceNumber)
      raises( InvalidResourceNumber );

  //## Operation: shutdown
  //## Documentation:
  //      This operation destroys all resources managed by
  //      this factory and terminates the factory server.
  void shutdown()
      raises( ShutdownFailure );



  //##begin ResourceFactory.additionalDeclarations preserve=yes
  //##end ResourceFactory.additionalDeclarations

};

//## Application Documentation:
//      The Application interface represents the
//      abstraction of an application executing within the
//      domain.  The interface provides the capabilities
//      for controlling an application within the radio.
//## Category: CF_IDL_Design_Components

interface Application : CF::Resource {
  //##begin Application.initialDeclarations preserve=yes
  //##end Application.initialDeclarations

  // Attributes

  //## Attribute: profile
  //## Documentation:
  //      This attribute is the XML profile information for
  //      the application.
  readonly attribute string profile;
  //## Attribute: name
  //## Documentation:
  //      This attribute is the name of the application.
  readonly attribute string name;

  // Relationships


  // Associations


  // Operations



  //##begin Application.additionalDeclarations preserve=yes
  //##end Application.additionalDeclarations

};
```

```
//## ApplicationFactory Documentation:
//      The Applicationfactory interface is used to create
//      applications within the domain.  Each Application
//      Factory object creates a specific application
//      (e.g., SINCGARS, LOS, Havequick, etc.).
//## Category: CF_IDL_Design_Components

interface ApplicationFactory {
  //##begin ApplicationFactory.initialDeclarations preserve=yes
  //##end ApplicationFactory.initialDeclarations

  // Nested Classes
  //## DeviceAssignmentType Documentation:
  //      The IDL structure, DeviceAssignmentType, provides
  //      the type to request execution of a Application
  //      component on a specific device.
  //## Category: CF_IDL_Design_Components

  struct DeviceAssignmentType {
    //##begin DeviceAssignmentType.initialDeclarations preserve=yes
    //##end DeviceAssignmentType.initialDeclarations

    // Attributes

    string componentID;
    string assignedDeviceID;

    // Relationships


    // Associations



    //##begin DeviceAssignmentType.additionalDeclarations preserve=yes
    //##end DeviceAssignmentType.additionalDeclarations

  };

  //## DeviceAssignmentSequence Documentation:
  //      The IDL sequence, DeviceAssignmentSequence,
  //      provides a unbounded sequence of 0..n of Device
  //      AssignmentType.
  //## Category: CF_IDL_Design_Components

  typedef sequence <DeviceAssignmentType> DeviceAssignmentSequence;


  //## CreateApplicationRequestError Documentation:
  //      The following IDL exception type is raised when the
  //      parameter DeviceAssignmentSequence contains one (1)
  //      or more invalid Application component to device
  //      assignment.
  //## Category: CF_IDL_Design_Components

  exception CreateApplicationRequestError {
```

```
      //##begin CreateApplicationRequestError.initialDeclarations preserve=yes
      //##end CreateApplicationRequestError.initialDeclarations

      // Attributes

      DeviceAssignmentSequence invalidAssignments;

      // Relationships


      // Associations



      //##begin CreateApplicationRequestError.additionalDeclarations
preserve=yes
      //##end CreateApplicationRequestError.additionalDeclarations

    };

    //## CreateApplicationError Documentation:
    //      The following IDL exception type is raised when the
    //      create request is valid but the Application is
    //      unsuccessfully instantiated due to internal
    //      processing errors.
    //## Category: CF_IDL_Design_Components

    exception CreateApplicationError {
      //##begin CreateApplicationError.initialDeclarations preserve=yes
      //##end CreateApplicationError.initialDeclarations

      // Attributes

      StringSequence errorMessages;

      // Relationships


      // Associations



      //##begin CreateApplicationError.additionalDeclarations preserve=yes
      //##end CreateApplicationError.additionalDeclarations

    };

    // Attributes

    //## Attribute: name
    //## Documentation:
    //      This attribute indicates the name of the Application
    //      Factory (e.g., SINCGARS, LOS, Havequick, DAMA25,
    //      etc.).
    readonly attribute string name;
    //## Attribute: softwareProfile
    //## Documentation:
```

```
    //      This attribute contains the application software
    //      profile that this factory uses when creating an
    //      application.
    readonly attribute string softwareProfile;

    // Relationships


    // Associations


    // Operations

    //## Operation: create
    //## Documentation:
    //      This operation is used to create an application
    //      within the radio domain.

    //      The createApplication operation provides a client
    //      interface to request the creation of an application
    //      on client requested device(s) or the creation of an
    //      application in which the ApplicationFactory
    //      determines the necessary device(s )in which to
    //      create the application.

    //      The following exception is raised by the create
    //      Application operation when the input device is
    //      invalid.
    //      exception CF::InvalidUUID

    //      The following exception is raised by the create
    //      Application operation when the input application is
    //      invalid.
    //      exception CF::InvalidUUID
    Application create(in string name, in Properties initConfiguration, in
DeviceAssignmentSequence deviceAssignments)
        raises( CreateApplicationRequestError, CreateApplicationError );



    //##begin ApplicationFactory.additionalDeclarations preserve=yes
    //##end ApplicationFactory.additionalDeclarations

  };

  //## Device Documentation:
  //      The Device interface defines the CORBA interfaces
  //      for communicating with a device.  A device contains
  //      state information and status attributes based upon
  //      its Device Profile definition.  The attributes that
  //      can be access or set are described in its Device
  //      Profile.

  //      The state information is based upon the X.731
  //      INFORMATION TECHNOLOGY - OPEN SYSTEMS
  //      INTERCONNECTION - SYSTEMS MANAGEMENT: STATE
  //      MANAGEMENT FUNCTION.  The identical text is also
```

```
//      published as ISO/IEC International Standard 10164-2.
//## Category: CF_IDL_Design_Components

interface Device : CF::Resource {
  //##begin Device.initialDeclarations preserve=yes
  //##end Device.initialDeclarations

  // Nested Classes
  //## InvalidProcess Documentation:
  //      This exception indicates that a process with that
  //      ID does not exist on this device.
  //## Category: CF_IDL_Design_Components

  exception InvalidProcess {
    //##begin InvalidProcess.initialDeclarations preserve=yes
    //##end InvalidProcess.initialDeclarations

    // Attributes


    // Relationships


    // Associations



    //##begin InvalidProcess.additionalDeclarations preserve=yes
    //##end InvalidProcess.additionalDeclarations

  };

  //## InvalidFunction Documentation:
  //      This exception indicates that a function with that
  //      name hasn't been loaded on this device.
  //## Category: CF_IDL_Design_Components

  exception InvalidFunction {
    //##begin InvalidFunction.initialDeclarations preserve=yes
    //##end InvalidFunction.initialDeclarations

    // Attributes


    // Relationships


    // Associations



    //##begin InvalidFunction.additionalDeclarations preserve=yes
    //##end InvalidFunction.additionalDeclarations

  };

  //## DeviceNotCapable Documentation:
```

```
//      This exception indicates the device is not capable
//      of this behavior (e.g., load, execute).
//## Category: CF_IDL_Design_Components

exception DeviceNotCapable {
  //##begin DeviceNotCapable.initialDeclarations preserve=yes
  //##end DeviceNotCapable.initialDeclarations

  // Attributes


  // Relationships


  // Associations



  //##begin DeviceNotCapable.additionalDeclarations preserve=yes
  //##end DeviceNotCapable.additionalDeclarations

};

//## InvalidCapacity Documentation:
//      This exception indicates the capacity is not valid
//      for this device.
//## Category: CF_IDL_Design_Components

exception InvalidCapacity {
  //##begin InvalidCapacity.initialDeclarations preserve=yes
  //##end InvalidCapacity.initialDeclarations

  // Attributes

  //## Attribute: msg
  //## Documentation:
  //      The message indicates the reason for the invalid
  //      capacity.
  string msg;

  // Relationships


  // Associations



  //##begin InvalidCapacity.additionalDeclarations preserve=yes
  //##end InvalidCapacity.additionalDeclarations

};

//## CapacityExceeded Documentation:
//      This exception indicates the capacity limits have
//      been exceeded.
//## Category: CF_IDL_Design_Components
```

```
exception CapacityExceeded {
  //##begin CapacityExceeded.initialDeclarations preserve=yes
  //##end CapacityExceeded.initialDeclarations

  // Attributes


  // Relationships


  // Associations



  //##begin CapacityExceeded.additionalDeclarations preserve=yes
  //##end CapacityExceeded.additionalDeclarations

};

//## AdminType Documentation:
//      This type is a CORBA IDL enumeratiuon type that
//      defines an object's administrative states.  The
//      administration state indicates the permission to
//      use or prohibition against using the resource.
//## Category: CF_IDL_Design_Components

enum AdminType
{
  LOCKED,
  SHUTTING_DOWN,
  UNLOCKED
};

//## OperationalType Documentation:
//      This type is a CORBA IDL enumeration type that
//      defines an object's  Operational states.   The
//      operational state indicates whether or not the
//      object is working or not.
//## Category: CF_IDL_Design_Components

enum OperationalType
{
  ENABLED,
  DISABLED
};

//## UsageType Documentation:
//      This type is a CORBA IDL enumeration type  that
//      defines the object's Usage states.   This state
//      indicates whether or not an object is actively in
//      use at a specific instant, and if so, whether or
//      not it has spare capacity for additional users at
//      that instant.
//## Category: CF_IDL_Design_Components

enum UsageType
{
```

```
  IDLE,
  ACTIVE,
  BUSY
};

//## ProcessID_Type Documentation:
//      This defines the process number within the radio.
//      Processor  number is only unique to the Processor
//      OS that created the process.
//## Category: CF_IDL_Design_Components

typedef unsigned long ProcessID_Type;


//## LoadType Documentation:
//      This type defines the type of load to be performed.
//      The loading of the software can be perform as a
//      driver, in the OS kernel memory, or as a
//      relocatable object.
//## Category: CF_IDL_Design_Components

enum LoadType
{
  KERNEL_MODULE,
  RELOCATABLE_OBJECT,
  DRIVER
};

// Attributes

//## Attribute: usageState
//## Documentation:
//      This state indicates whether or not an device is
//      actively in use at a specific instant, and if so,
//      whether or not it has spare capacity for additional
//      users at that instant.
readonly attribute UsageType usageState;
//## Attribute: adminState
//## Documentation:
//      The administration state indicates the permission
//      to use or prohibition against using the device.
attribute AdminType adminState;
//## Attribute: operationalState
//## Documentation:
//      The operational state indicates whether or not the
//      Device is working or not.
readonly attribute OperationalType operationalState;
//## Attribute: identifier
//## Documentation:
//      This attribute is the unique identifier for a
//      device instance.
readonly attribute string identifier;
//## Attribute: softwareProfile
//## Documentation:
//      This attribute defines the logical device driver
//      XML profile capabilities.
readonly attribute string softwareProfile;
```

```
//## Attribute: label
//## Documentation:
//      This attribute attribute is the physical location
//      label for this device.
readonly attribute string label;
//## Attribute: parentDevice
//## Documentation:
//      This attribute indicates the parent device this
//      device is assocated with by either being a part of
//      or was created from.
readonly attribute Device parentDevice;
//## Attribute: devices
//## Documentation:
//      This attribute provides a list of the hardware
//      devices along with their properties that are
//      currently associated with this Device object.
readonly attribute DeviceSequence devices;

// Relationships


// Associations


// Operations

//## Operation: terminate
//## Documentation:
//      The terminate operation terminates the execution of
//      the function on the device.  This operation raises
//      InvalidProcess when the processID does not exist
//      for that device.  The operation also raises Device
//      NotCapable when the device is not capable of this
//      behavior.
void terminate(in ProcessID_Type processId)
    raises( InvalidProcess, DeviceNotCapable );

//## Operation: execute
//## Documentation:
//      The execute operation executes the given function
//      name using the arguments that have been passed in
//      and returns an ID of the process that has been
//      created.

//      This operation raises DeviceNotCapable exception
//      when the device is not capable of executing
//      software.  This operation raises InvalidFunction
//      exception when the function does not exists which
//      means it hasn't been loaded on that device.
ProcessID_Type execute(in string functionName, in Properties parameters)
    raises( InvalidFunction, DeviceNotCapable );

//## Operation: executeProcess
//## Documentation:
//      The execute operation executes the given file name
//      using the arguments that have been passed in and
//      returns an ID of the process that has been
```

```
    //      created.   If the input FileSystem is nil, then the
    //      operation uses the DeviceManager's FileManager for
    //      finding the file to execute.

    //      This operation raises DeviceNotCapable exception
    //      when the device is not capable of executing
    //      software.  This operation raises InvalidFunction
    //      exception when the function does not exists which
    //      means it hasn't been loaded on that device.
    ProcessID_Type executeProcess(in FileSystem fs, in string fileName, in
Properties parameters)
        raises( InvalidFileName, DeviceNotCapable );

    //## Operation: load
    //## Documentation:
    //      The load operation loads a file on the specified
    //      device based on the given loadKind and fileName
    //      using the input FileSystem to retrieve it.  If the
    //      input FileSystem is nil, then the operation uses
    //      the DeviceManager's FileManager for finding the
    //      file to loaded.

    //      This operationl raises DeviceNotCapable exception
    //      when the device is not capable of loading the
    //      application (e.g., serial, audio, ethernet, etc.).
    //      This operation raises CF::InvalidFileName exception
    //      when the file does not exist.
    void load(in FileSystem fs, in string fileName, in LoadType loadKind)
        raises( DeviceNotCapable, InvalidFileName );

    //## Operation: unload
    //## Documentation:
    //      The unload operation unloads application software
    //      on the specified device based on the input file
    //      Name.  This operation raises DeviceNotCapable
    //      exception when the device is not capable of
    //      unloading software (e.g., serial, audio, ethernet,
    //      etc.).  This operation raises CF::InvalidFileName
    //      exception when the file does not exist.
    void unload(in string fileName)
        raises( DeviceNotCapable, InvalidFileName );

    //## Operation: allocateCapacity
    //## Documentation:
    //      This operation requests capacity from the device.
    //      The current capacity is decremented by the input
    //      capacity amount based upon the capacity model.  The
    //      operation returns the available capacity. The
    //      CapacityExceeded exception is raised when the
    //      capacity cannot be granted. The InvalidCapacity
    //      exception is raised when the capacity does not
    //      exist or the capacity value is invalid for this
    //      Device.
    void allocateCapacity(inout DataType capacity)
        raises( CapacityExceeded, InvalidCapacity );

    //## Operation: deallocateCapacity
```

```
  //## Documentation:
  //      This operation increments the capacity by the input
  //      capacity amount.  The operation raises Invalid
  //      Capacity when the input capacity ID is invalid.
  void deallocateCapacity(in DataType capacity)
      raises( InvalidCapacity );

  //## Operation: addDevice
  //## Documentation:
  //      This operation adds a device to a parent device.  A
  //      given Device may have many Devices it is
  //      responsible for or is associated with.
  void addDevice(in Device associatedDevice);

  //## Operation: removeDevice
  //## Documentation:
  //      This operation removes a device associated with
  //      another device.
  void removeDevice(in Device associatedDevice);



  //##begin Device.additionalDeclarations preserve=yes
  //##end Device.additionalDeclarations

};

//## DeviceManager Documentation:
//      The DeviceManager interface defines the CORBA
//      interfaces for communicating with a device that is
//      CORBA capable.  A DeviceManager object dynamically
//      receives load and execute requests.

//      A DeviceManager upon startup determines its local
//      devices and may create or obtain a  Logger and File
//      System objects.  The relatiuonships for this
//      interface are shown in the DeviceManager
//      Relationships figure.
//## Category: CF_IDL_Design_Components

interface DeviceManager : CF::Device {
  //##begin DeviceManager.initialDeclarations preserve=yes
  //##end DeviceManager.initialDeclarations

  // Attributes

  readonly attribute string deviceManagerProfile;
  //## Attribute: log
  //## Documentation:
  //      The Log attribute is the logger associated with
  //      this device.
  readonly attribute Logger log;
  //## Attribute: fileMgr
  //## Documentation:
  //      The FileMgr attribute is the CF::FileManager object
  //      associated with this device.
  readonly attribute FileManager fileMgr;
```

```
    // Relationships


    // Associations


    // Operations

    void installDevice(in FileSystem hardwareFS, in string
deviceProfileFileName, in FileSystem softwareFS, in string swProfileFileName)
        raises( InvalidFileName, InvalidProfile );



    //##begin DeviceManager.additionalDeclarations preserve=yes
    //##end DeviceManager.additionalDeclarations

  };

  //## DomainManager Documentation:
  //      The DomainManager interface API is for the control
  //      and configuration of the radio domain.

  //      The DomainManager interface can be logically
  //      grouped into three categories: Human Computer
  //      Interface (HCI), Registration, and Core Framework
  //      (CF) administration.
  //      1. The HCI operations are used to configure the
  //      domain, get the domain's capabilities (devices and
  //      applications), query the domain, and initiate
  //      maintenance functions.  Host operations are
  //      performed by a client user interface capable of
  //      interfacing to the Domain Manager.
  //      2. The registration operations are used to register
  //      / unregister DeviceManagers and DeviceManager's
  //      devices and applications at startup or during
  //      run-time for dynamic device and application
  //      extraction and insertion.
  //      3. The administration operations are used to access
  //      the interfaces of registered DeviceManagers, File
  //      Managers, and Loggers of the domain.
  //## Category: CF_IDL_Design_Components

  interface DomainManager {
    //##begin DomainManager.initialDeclarations preserve=yes
    //##end DomainManager.initialDeclarations

    // Nested Classes
    //## ApplicationInstallationError Documentation:
    //      This exception indicates an application
    //      installation has not completed correctly.
    //## Category: CF_IDL_Design_Components

    exception ApplicationInstallationError {
      //##begin ApplicationInstallationError.initialDeclarations preserve=yes
      //##end ApplicationInstallationError.initialDeclarations
```

```
      // Attributes


      // Relationships


      // Associations



      //##begin ApplicationInstallationError.additionalDeclarations
preserve=yes
      //##end ApplicationInstallationError.additionalDeclarations

    };

    //## ApplicationSequence Documentation:
    //      This type defines an unbounded sequence of
    //      Applications.
    //## Category: CF_IDL_Design_Components

    typedef sequence <Application> ApplicationSequence;


    //## ApplicationFactorySequence Documentation:
    //      This type defines an unbounded sequence of
    //      application factories.
    //## Category: CF_IDL_Design_Components

    typedef sequence <ApplicationFactory> ApplicationFactorySequence;


    //## DeviceManagerSequence Documentation:
    //      This type defines an unbounded sequence of device
    //      managers.
    //## Category: CF_IDL_Design_Components

    typedef sequence <DeviceManager> DeviceManagerSequence;


    //## InvalidIdentifier Documentation:
    //      This exception indicates the application ID is
    //      invalid.
    //## Category: CF_IDL_Design_Components

    exception InvalidIdentifier {
      //##begin InvalidIdentifier.initialDeclarations preserve=yes
      //##end InvalidIdentifier.initialDeclarations

      // Attributes


      // Relationships


      // Associations
```

```
  //##begin InvalidIdentifier.additionalDeclarations preserve=yes
  //##end InvalidIdentifier.additionalDeclarations

};

// Attributes

//## Attribute: deviceManagers
//## Documentation:
//      This attribute is a list of the DeviceManagers
//      within the domain.
readonly attribute DeviceManagerSequence deviceManagers;
//## Attribute: applications
//## Documentation:
//      This attribute is a list of unbounded sequence of
//      applications.
readonly attribute ApplicationSequence applications;
//## Attribute: applicationFactories
//## Documentation:
//      This attribute is an unbounded sequence of
//      application factories.
readonly attribute ApplicationFactorySequence applicationFactories;
//## Attribute: fileMgr
//## Documentation:
//      This attribute indicates the FileManager for the
//      Domain.
readonly attribute FileManager fileMgr;

// Relationships


// Associations


// Operations

//## Operation: registerDevice
//## Documentation:
//      This operation is used to register a Device for a
//      specific DeviceManager in the DomainManager's
//      Domain Profile.

//      The following exception is raised by the register
//      Device operation when the input DeviceManagerID
//      does not match the ID of any registered Device
//      Manager.
//      exception CF::InvalidUUID

//      The following exception is raised by the register
//      Device operation when the input deviceProfile is
//      invalid.
//      exception CF::InvalidProfile
void registerDevice(in Device registeringDevice)
    raises( InvalidObjectReference, InvalidProfile );
```

```
//## Operation: registerDeviceManager
//## Documentation:
//      This operation registers a DeviceManager, the Device
//      Manager's device(s), and application(s) in the
//      DomainManager's Domain Profile.

//      The following exception is raised by the register
//      DeviceManager operation when input parameter Device
//      Manager contains an invalid reference to a Device
//      Manager interface.
//      exception InvalidObjectReference
void registerDeviceManager(in DeviceManager deviceMgr)
    raises( InvalidObjectReference, InvalidProfile );

//## Operation: unregisterDeviceManager
//## Documentation:
//      This operation is used to unregister a Device
//      Manager object from the DomainManager's Domain
//      Profile.  A DeviceManager may be unregistered
//      during run-time for dynamic extraction or
//      maintenance of the DeviceManager.  The following
//      exception is raised by the unregisterDeviceManager
//      operation when input parameter DeviceManagerID is
//      not a registered DeviceManager in the Domain
//      Profile.
//      exception   CF::InvalidUUID
void unregisterDeviceManager(in DeviceManager deviceMgr)
    raises( InvalidObjectReference );

//## Operation: unregisterDevice
//## Documentation:
//      This operation is used to remove a device entry
//      from the DomainManager for a specific DeviceManager.

//      The following exception is raised by the unregister
//      Device operation when the input parameter Device
//      ManagerID is not a registered with the Domain
//      Manager.
//      exception CF::InvalidUUID

//      The following exception is raised by the unregister
//      Device operation when the input parameter deviceID
//      is not valid device of the registered DeviceManager.
//      exception CF::InvalidUUID
void unregisterDevice(in Device unregisteringDevice)
    raises( InvalidObjectReference );

//## Operation: installApplication
//## Documentation:
//      This operation is used to register new application
//      software in the DomainManager's Domain Profile.

//      The CF Installer typically invokes this operation
//      when it has completed the installation of a new
//      application into the radio domain.
```

```
    //      The following exception is raised by the register
    //      Application operation when the input fileProfile is
    //      invalid.
    //      exception CF::InvalidProfile

    //      The following exception is raised by the register
    //      Application operation when the input application
    //      profile file name is invalid.
    //      exception CF::InvalidFileName

    //      The following exception is raised by the register
    //      Application operation when the installation of the
    //      Application files was not completed correctly.
    //      exception ApplicationRegistrationError
    void installApplication(in string profileFileName)
        raises( InvalidProfile, InvalidFileName, ApplicationInstallationError
);

    //## Operation: uninstallApplication
    //## Documentation:
    //      This operation is used to uninstall an application
    //      in the DomainManager's Domain Profile.
    //      The CF Installer typically invokes this operation
    //      when removing an application from the radio domain.

    //      The following exception is raised by the uninstall
    //      Application operation when the applicationID is
    //      invalid.
    //      exception  CF::InvalidIdentifier
    void uninstallApplication(in string applicationID)
        raises( InvalidIdentifier );



    //##begin DomainManager.additionalDeclarations preserve=yes
    //##end DomainManager.additionalDeclarations

  };

};

#endif
```

C.2 PortTypes MODULE.

This CORBA Module contains a set of unbundled CORBA sequence types based on CORBA types not in the CF CORBA Module. The Basic Sequence Types IDL was generated from the Rational Rose model.

```
//## Module: PortTypes
//## Subsystem:
Core_CSCI_IDL_Implementation_Components::CF_IDL_Implementation_Component
//## Source file:
/software/components/RadioCORBA/mmits_sdr/sdr_code.ss/glbick.wrk/PortTypes.idl
//##begin module.cm preserve=no
//    %X% %Q% %Z% %W%
//##end module.cm

//##begin module.cp preserve=no
//##end module.cp


#ifndef PortTypes_idl
#define PortTypes_idl

//##begin module.additionalIncludes preserve=no
//##end module.additionalIncludes

//##begin module.includes preserve=yes
//##end module.includes

#include "CF.idl"

// ============================================================================

module PortTypes
{
  //##begin module.declarations preserve=no
  //##end module.declarations

  //##begin module.additionalDeclarations preserve=yes
  //##end module.additionalDeclarations


  //## WstringSequence Documentation:
  //      This type is a CORBA unbounded sequence of Wstrings.
  //## Category: Port_Types_IDL_Components

// Unsupported by ORBexpress at this time
//  typedef sequence<wstring> WstringSequence;


  //## BooleanSequence Documentation:
  //      This type is a CORBA unbounded sequence of booleans.
  //## Category: Port_Types_IDL_Components

  typedef sequence<boolean> BooleanSequence;
```

```
  //## CharSequence Documentation:
  //      This type is a CORBA unbounded sequence of
  //      characters.
  //## Category: Port_Types_IDL_Components

  typedef sequence<char> CharSequence;


  //## DoubleSequence Documentation:
  //      This type is a CORBA unbounded sequence of doubles.
  //## Category: Port_Types_IDL_Components

  typedef sequence<double> DoubleSequence;


  //## LongDoubleSequence Documentation:
  //      This type is a CORBA unbounded sequence of long
  //      Doubles.
  //## Category: Port_Types_IDL_Components

// Unsupported by ORBexpress at this time
//  typedef sequence<long double> LongDoubleSequence;


  //## LongLongSequence Documentation:
  //      This type is a CORBA unbounded sequence of
  //      longlongs.
  //## Category: Port_Types_IDL_Components

// Unsupported by ORBexpress at this time
//  typedef sequence<long long> LongLongSequence;


  //## LongSequence Documentation:
  //      This type is a CORBA unbounded sequence of longs.
  //## Category: Port_Types_IDL_Components

  typedef sequence<long> LongSequence;


  //## ShortSequence Documentation:
  //      This type is a CORBA unbounded sequence of shorts.
  //## Category: Port_Types_IDL_Components

  typedef sequence<short> ShortSequence;


  //## UlongLongSequence Documentation:
  //      This type is a CORBA unbounded sequence of unsigned
  //      lon longs.
  //## Category: Port_Types_IDL_Components

// Unsupported by ORBexpress at this time
//  typedef sequence<unsigned long long> UlongLongSequence;
```

```
  //## UlongSequence Documentation:
  //      This type is a CORBA unbounded sequence of unsigned
  //      longs.
  //## Category: Port_Types_IDL_Components

  typedef sequence<unsigned long> UlongSequence;


  //## UshortSequence Documentation:
  //      This type is a CORBA unbounded sequence of unsigned
  //      shorts.
  //## Category: Port_Types_IDL_Components

  typedef sequence<unsigned short> UshortSequence;


  //## WcharSequence Documentation:
  //      This type is a CORBA unbounded sequence of
  //      wcharacters.
  //## Category: Port_Types_IDL_Components

// Unsupported by ORBexpress at this time
//  typedef sequence<wchar> WcharSequence;


  //## FloatSequence Documentation:
  //      This type is a CORBA unbounded sequence of floats.
  //## Category: Port_Types_IDL_Components

  typedef sequence<float> FloatSequence;


};

#endif
```

C.3 PushPorts MODULE.

This CORBA Module contains the PushPorts interfaces where each interface is based upon one CORBA basic type. The PushPorts CORBA module contains a set of data interfaces that extend the CF::*Port* interface. Each interface is a push consumer type where a producer uses that interface to push data to a consumer. Each interface contains one "one-way" operation that is based upon a standard CORBA type such as octet, short, and long. The operation parameters are a sequence of a basic CORBA type parameter along with a control parameter. The types for these parameters are defined in PortTypes CORBA module. The requirements for these interfaces are implementation-dependent and the control information is port interface dependent. The PushPorts CORBA module can be implemented either as a CORBA TIE or non-TIE approach. The benefit of the TIE approach is it allows a push consumer servant to implement more than one of these interfaces, which is allowed by the Software Profile. This standard set of interfaces can be used by application developers for defining their CF::*Port* interfaces. The module diagram for PushPorts is shown in Figure C-3.

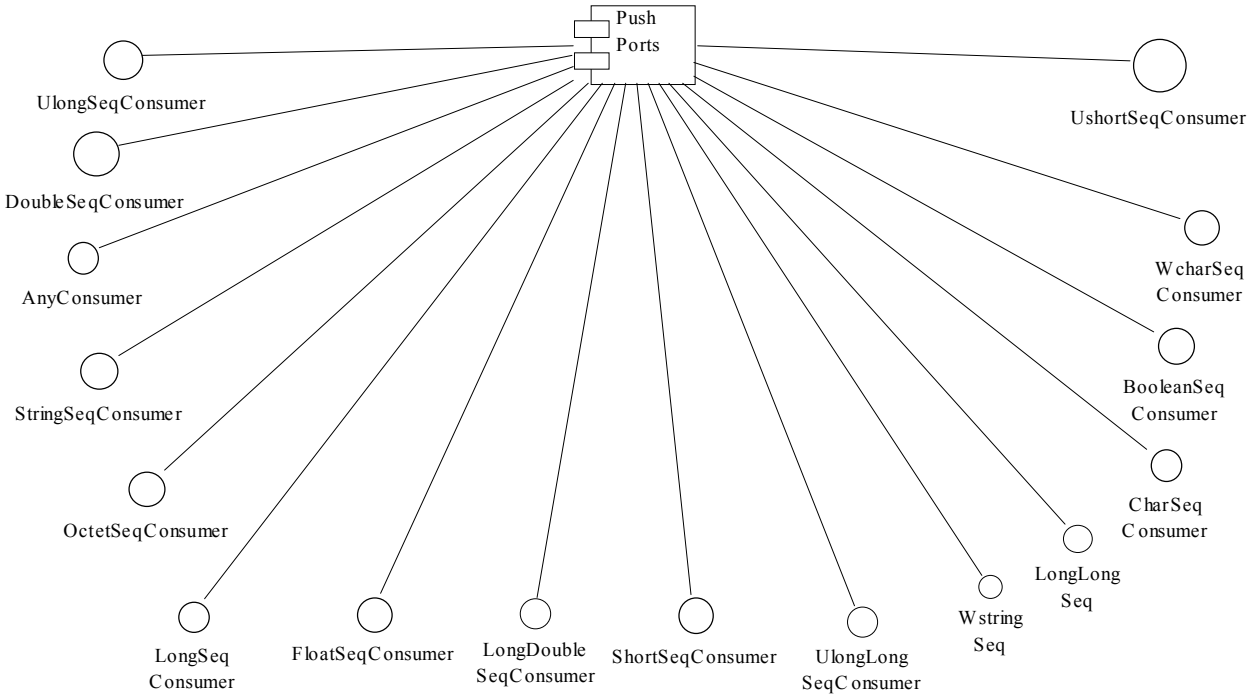

Figure C-3. PushPorts

The following is the PushPorts IDL generated from the Rational Rose model.

```
//## Module: PushPorts
//## Subsystem:
Core_CSCI_IDL_Implementation_Components::CF_IDL_Implementation_Component
```

```
//## Source file:
/software/components/RadioCORBA/mmits_sdr/sdr_code.ss/glbick.wrk/PushPorts.idl
//## Documentation::
//      This CORBA Module contains the Push Port interfaces
//      where each interface is based upon one CORBA basic
//      type.
//##begin module.cm preserve=no
//    %X% %Q% %Z% %W%
//##end module.cm

//##begin module.cp preserve=no
//##end module.cp


#ifndef PushPorts_idl
#define PushPorts_idl

//##begin module.additionalIncludes preserve=no
//##end module.additionalIncludes

//##begin module.includes preserve=yes
//##end module.includes

#include "CF.idl"
#include "PortTypes.idl"

// ============================================================

module PushPorts
{
  //##begin module.declarations preserve=no
  //##end module.declarations

  //##begin module.additionalDeclarations preserve=yes
  //##end module.additionalDeclarations


  //## OctetSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process an octet sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface OctetSeqConsumer : CF::Port {
    //##begin OctetSeqConsumer.initialDeclarations preserve=yes
    //##end OctetSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations
```

```
    //## Operation: processOctetMsg
    //## Documentation:
    //      This operation is used to push a sequence of Octets
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushConsumer) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processOctetMsg(in CF::OctetSequence msg, in CF::Properties
options);



    //##begin OctetSeqConsumer.additionalDeclarations preserve=yes
    //##end OctetSeqConsumer.additionalDeclarations

  };

  //## WcharSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a wide character sequence pushed to
  //      them by producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

//  interface WcharSeqConsumer : CF::Port {
    //##begin WcharSeqConsumer.initialDeclarations preserve=yes
    //##end WcharSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processWcharMsg
    //## Documentation:
    //      This operation is used to push a sequence of Wchars
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
// Unsupported by ORBexpress at this time
//    oneway void processWcharMsg(in PortTypes::WcharSequence msg, in
CF::Properties options);



    //##begin WcharSeqConsumer.additionalDeclarations preserve=yes
    //##end WcharSeqConsumer.additionalDeclarations
```

```
//  };

  //## LongSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a long sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface LongSeqConsumer : CF::Port {
    //##begin LongSeqConsumer.initialDeclarations preserve=yes
    //##end LongSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processLongMsg
    //## Documentation:
    //      This operation is used to push a sequence of Longs
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processLongMsg(in PortTypes::LongSequence msg, in
CF::Properties options);



    //##begin LongSeqConsumer.additionalDeclarations preserve=yes
    //##end LongSeqConsumer.additionalDeclarations

  };

  //## ShortSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a short sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface ShortSeqConsumer : CF::Port {
    //##begin ShortSeqConsumer.initialDeclarations preserve=yes
    //##end ShortSeqConsumer.initialDeclarations

    // Attributes


    // Relationships
```

```
    // Associations


    // Operations

    //## Operation: processShortMsg
    //## Documentation:
    //      This operation is used to push a sequence of Shorts
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processShortMsg(in PortTypes::ShortSequence msg, in
CF::Properties options);



    //##begin ShortSeqConsumer.additionalDeclarations preserve=yes
    //##end ShortSeqConsumer.additionalDeclarations

  };

  //## LongLongSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a CORBA long long sequence pushed to
  //      them by producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface LongLongSeqConsumer : CF::Port {
    //##begin LongLongSeqConsumer.initialDeclarations preserve=yes
    //##end LongLongSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processLongLongMsg
    //## Documentation:
    //      This operation is used to push a sequence of Long
    //      Longs information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PushSource) object. The message
    //      being pushed has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
// Unsupported by ORBexpress at this time
//    oneway void processLongLongMsg(in PortTypes::LongLongSequence msg, in
CF::Properties options);
```

```
    //##begin LongLongSeqConsumer.additionalDeclarations preserve=yes
    //##end LongLongSeqConsumer.additionalDeclarations

  };

  //## UlongSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process an unsigned long sequence pushed to
  //      them by producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface UlongSeqConsumer : CF::Port {
    //##begin UlongSeqConsumer.initialDeclarations preserve=yes
    //##end UlongSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processUlongMsg
    //## Documentation:
    //      This operation is used to push a sequence of
    //      Unsigned Longs information to be received or
    //      transmitted through the RADIO from one object to
    //      the next "destination" (PushSource) object. The
    //      message being pushed has data and control
    //      information (classification, source, destination,
    //      priority, etc.).
    oneway void processUlongMsg(in PortTypes::UlongSequence msg, in
CF::Properties options);



    //##begin UlongSeqConsumer.additionalDeclarations preserve=yes
    //##end UlongSeqConsumer.additionalDeclarations

  };

  //## UlongLongSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process an unsigned long long sequence pushed
  //      to them by producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface UlongLongSeqConsumer : CF::Port {
    //##begin UlongLongSeqConsumer.initialDeclarations preserve=yes
    //##end UlongLongSeqConsumer.initialDeclarations
```

```
    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processULongLongMsg
    //## Documentation:
    //      This operation is used to push a sequence of
    //      Unsigned Long Longs information to be received or
    //      transmitted through the RADIO from one object to
    //      the next "destination" (PushSource) object. The
    //      message being pushed has data and control
    //      information (classification, source, destination,
    //      priority, etc.).
// Unsupported by ORBexpress at this time
//    oneway void processULongLongMsg(in PortTypes::UlongLongSequence msg, in
CF::Properties options);



    //##begin UlongLongSeqConsumer.additionalDeclarations preserve=yes
    //##end UlongLongSeqConsumer.additionalDeclarations

  };

  //## FloatSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a float sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface FloatSeqConsumer : CF::Port {
    //##begin FloatSeqConsumer.initialDeclarations preserve=yes
    //##end FloatSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processFloatMsg
    //## Documentation:
    //      This operation is used to push a sequence of floats
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
```

```
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processFloatMsg(in PortTypes::FloatSequence msg, in
CF::Properties options);



    //##begin FloatSeqConsumer.additionalDeclarations preserve=yes
    //##end FloatSeqConsumer.additionalDeclarations

  };

  //## DoubleSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a double sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface DoubleSeqConsumer : CF::Port {
    //##begin DoubleSeqConsumer.initialDeclarations preserve=yes
    //##end DoubleSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processDoubleMsg
    //## Documentation:
    //      This operation is used to push a sequence of
    //      Doubles information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PushSource) object. The message
    //      being pushed has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
    oneway void processDoubleMsg(in PortTypes::DoubleSequence msg, in
CF::Properties options);



    //##begin DoubleSeqConsumer.additionalDeclarations preserve=yes
    //##end DoubleSeqConsumer.additionalDeclarations

  };

  //## LongDoubleSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a long double sequence pushed to them
  //      by producers.
```

```
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface LongDoubleSeqConsumer : CF::Port {
    //##begin LongDoubleSeqConsumer.initialDeclarations preserve=yes
    //##end LongDoubleSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processLongDoubleMsg
    //## Documentation:
    //      This operation is used to push a sequence of Long
    //      Doubles information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PushSource) object. The message
    //      being pushed has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
// Unsupported by ORBexpress at this time
//    oneway void processLongDoubleMsg(in PortTypes::LongDoubleSequence msg, in
CF::Properties options);



    //##begin LongDoubleSeqConsumer.additionalDeclarations preserve=yes
    //##end LongDoubleSeqConsumer.additionalDeclarations

  };

  //## BooleanSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a boolean sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface BooleanSeqConsumer : CF::Port {
    //##begin BooleanSeqConsumer.initialDeclarations preserve=yes
    //##end BooleanSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations
```

```
    //## Operation: processBooleanMsg
    //## Documentation:
    //      This operation is used to push a sequence of
    //      Booleans information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PushSource) object. The message
    //      being pushed has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
    oneway void processBooleanMsg(in PortTypes::BooleanSequence msg, in
CF::Properties options);



    //##begin BooleanSeqConsumer.additionalDeclarations preserve=yes
    //##end BooleanSeqConsumer.additionalDeclarations

  };

  //## CharSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a character sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface CharSeqConsumer : CF::Port {
    //##begin CharSeqConsumer.initialDeclarations preserve=yes
    //##end CharSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processCharMsg
    //## Documentation:
    //      This operation is used to push a sequence of Chars
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processCharMsg(in PortTypes::CharSequence msg, in
CF::Properties options);



    //##begin CharSeqConsumer.additionalDeclarations preserve=yes
    //##end CharSeqConsumer.additionalDeclarations
```

```
  };

  //## UshortSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process an unsigned short sequence pushed to
  //      them by producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface UshortSeqConsumer : CF::Port {
    //##begin UshortSeqConsumer.initialDeclarations preserve=yes
    //##end UshortSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processUshortMsg
    //## Documentation:
    //      This operation is used to push a sequence of
    //      Unsigned Shorts information to be received or
    //      transmitted through the RADIO from one object to
    //      the next "destination" (PushSource) object. The
    //      message being pushed has data and control
    //      information (classification, source, destination,
    //      priority, etc.).
    oneway void processUshortMsg(in PortTypes::UshortSequence msg, in
CF::Properties options);



    //##begin UshortSeqConsumer.additionalDeclarations preserve=yes
    //##end UshortSeqConsumer.additionalDeclarations

  };

  //## StringSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a string sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface StringSeqConsumer : CF::Port {
    //##begin StringSeqConsumer.initialDeclarations preserve=yes
    //##end StringSeqConsumer.initialDeclarations

    // Attributes


    // Relationships
```

```
    // Associations


    // Operations

    //## Operation: processStringMsg
    //## Documentation:
    //      This operation is used to push a CORBA string
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processStringMsg(in CF::StringSequence msg, in CF::Properties
options);



    //##begin StringSeqConsumer.additionalDeclarations preserve=yes
    //##end StringSeqConsumer.additionalDeclarations

  };

  //## WstringSeqConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process a wide string sequence pushed to them
  //      by producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

//  interface WstringSeqConsumer : CF::Port {
    //##begin WstringSeqConsumer.initialDeclarations preserve=yes
    //##end WstringSeqConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processWstringMsg
    //## Documentation:
    //      This operation is used to push a CORBA Wstring
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushSource) object. The message being pushed has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
// Unsupported by ORBexpress at this time
//    void processWstringMsg(in PortTypes::WstringSequence msg, in
CF::Properties options);
```

```
    //##begin WstringSeqConsumer.additionalDeclarations preserve=yes
    //##end WstringSeqConsumer.additionalDeclarations

//  };

  //## AnyConsumer Documentation:
  //      This interface is implemented by push consumers
  //      that process an any sequence pushed to them by
  //      producers.
  //## Category: Push_Port_Consumer_IDL_Design_Components

  interface AnyConsumer : CF::Port {
    //##begin AnyConsumer.initialDeclarations preserve=yes
    //##end AnyConsumer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: processMsg
    //## Documentation:
    //      This operation is used to push a CORBA any
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PushConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    oneway void processMsg(in CF::DataType msg, in CF::Properties options);



    //##begin AnyConsumer.additionalDeclarations preserve=yes
    //##end AnyConsumer.additionalDeclarations

  };

};

#endif
```

C.4 PullPorts MODULE.

This CORBA Module contains the PullPorts interfaces where each interface is based upon one CORBA basic type. The PullPorts CORBA module contains a set of data interfaces that extend the CF::*Port* interface. Each interface is a pull producer type where a consumer uses that interface to pull data from a producer. Each interface contains one "two-way" operation which is based upon a standard CORBA type such as octet, short, and long. The operation "out" parameters are a sequence of a basic CORBA type parameter along with a control parameter. The types for these parameters are defined in PortTypes CORBA module. The requirements for these interfaces are implementation-dependent and the control information is port interface dependent. The PullPorts CORBA module can be implemented either as a CORBA TIE or non-TIE approach. This standard set of interfaces can be used by application developers for defining their CF::*Port* interfaces. The module diagram for PullPorts is shown in Figure C-4.

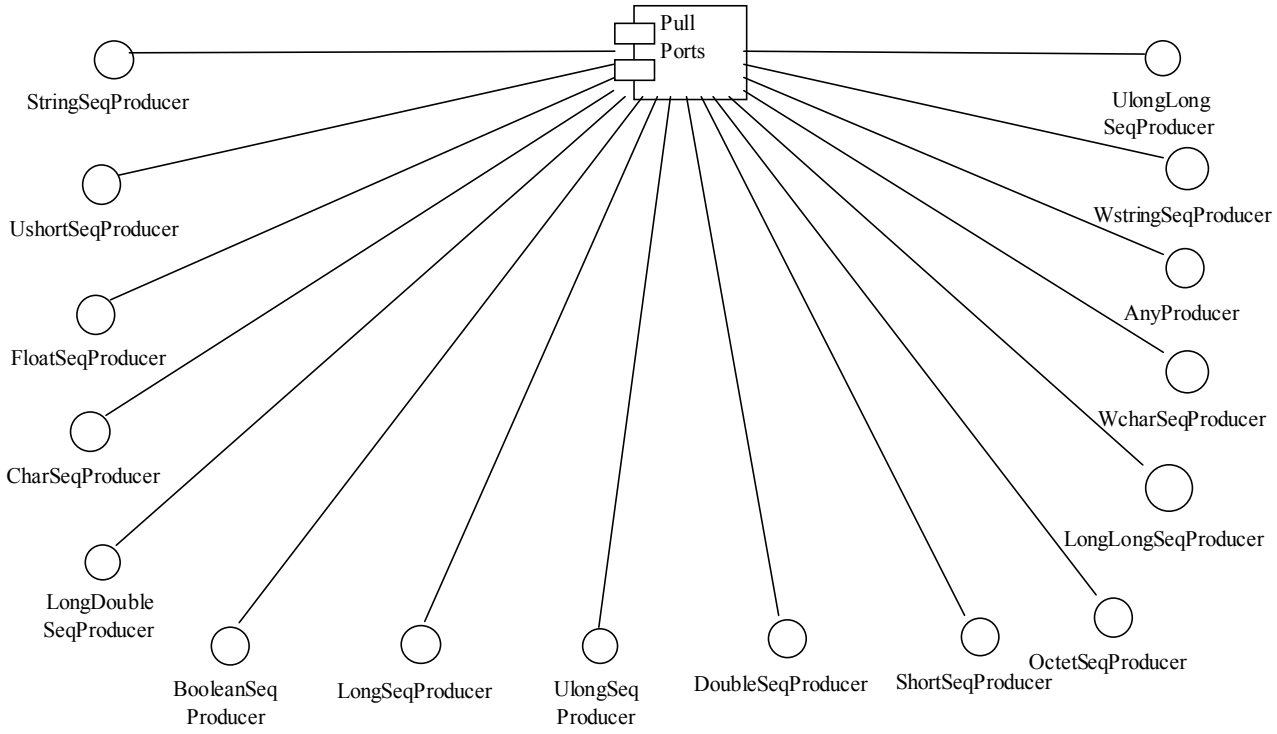

Figure C-4. PullPorts

The following is the PullPorts IDL generated from the Rational Rose model.

```
//## Module: PullPorts
//## Subsystem:
Core_CSCI_IDL_Implementation_Components::CF_IDL_Implementation_Component
```

```
//## Source file:
/software/components/RadioCORBA/mmits_sdr/sdr_code.ss/glbick.wrk/PullPorts.idl
//## Documentation::
//      This CORBA Module contains the Pul Port interfaces
//      where each interface is based upon one CORBA basic
//      type.
//##begin module.cm preserve=no
//    %X% %Q% %Z% %W%
//##end module.cm

//##begin module.cp preserve=no
//##end module.cp


#ifndef PullPorts_idl
#define PullPorts_idl

//##begin module.additionalIncludes preserve=no
//##end module.additionalIncludes

//##begin module.includes preserve=yes
//##end module.includes

#include "CF.idl"
#include "PortTypes.idl"

// ============================================================

module PullPorts
{
  //##begin module.declarations preserve=no
  //##end module.declarations

  //##begin module.additionalDeclarations preserve=yes
  //##end module.additionalDeclarations


  //## OctetSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) an octet
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface OctetSeqProducer : CF::Port {
    //##begin OctetSeqProducer.initialDeclarations preserve=yes
    //##end OctetSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations
```

```
    //## Operation: getOctetMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Octets
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    void getOctetMsg(out CF::OctetSequence msg, out CF::Properties options);



    //##begin OctetSeqProducer.additionalDeclarations preserve=yes
    //##end OctetSeqProducer.additionalDeclarations

  };

  //## WcharSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a wide
  //      character sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

//  interface WcharSeqProducer : CF::Port {
    //##begin WcharSeqProducer.initialDeclarations preserve=yes
    //##end WcharSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getWcharMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Wchars
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
// Unsupported by ORBexpress at this time
//    void getWcharMsg(out PortTypes::WcharSequence msg, out CF::Properties
options);



    //##begin WcharSeqProducer.additionalDeclarations preserve=yes
    //##end WcharSeqProducer.additionalDeclarations

//  };
```

```
  //## LongSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a long
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface LongSeqProducer : CF::Port {
    //##begin LongSeqProducer.initialDeclarations preserve=yes
    //##end LongSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getLongMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Longs
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    void getLongMsg(out PortTypes::LongSequence msg, out CF::Properties
options);



    //##begin LongSeqProducer.additionalDeclarations preserve=yes
    //##end LongSeqProducer.additionalDeclarations

  };

  //## FloatSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a float
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface FloatSeqProducer : CF::Port {
    //##begin FloatSeqProducer.initialDeclarations preserve=yes
    //##end FloatSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations
```

```
    // Operations

    //## Operation: getFloatMsg
    //## Documentation:
    //      This operation is used to pull a sequence of floats
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    void getFloatMsg(out PortTypes::FloatSequence msg, out CF::Properties
options);



    //##begin FloatSeqProducer.additionalDeclarations preserve=yes
    //##end FloatSeqProducer.additionalDeclarations

  };

  //## DoubleSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a double
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface DoubleSeqProducer : CF::Port {
    //##begin DoubleSeqProducer.initialDeclarations preserve=yes
    //##end DoubleSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getDoubleMsg
    //## Documentation:
    //      This operation is used to pull a sequence of
    //      Doubles information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PullConsumer) object. The message
    //      being pulled has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
    void getDoubleMsg(out PortTypes::DoubleSequence msg, out CF::Properties
options);
```

```
    //##begin DoubleSeqProducer.additionalDeclarations preserve=yes
    //##end DoubleSeqProducer.additionalDeclarations

  };

  //## LongDoubleSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a long
  //      double sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

//  interface LongDoubleSeqProducer : CF::Port {
    //##begin LongDoubleSeqProducer.initialDeclarations preserve=yes
    //##end LongDoubleSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getLongDoubleMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Long
    //      Doubles information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PullConsumer) object. The message
    //      being pulled has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
// Unsupported by ORBexpress at this time
//    void getLongDoubleMsg(out PortTypes::LongDoubleSequence msg, out
CF::Properties options);



    //##begin LongDoubleSeqProducer.additionalDeclarations preserve=yes
    //##end LongDoubleSeqProducer.additionalDeclarations

//  };

  //## WstringSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a wide
  //      string sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

//  interface WstringSeqProducer : CF::Port {
    //##begin WstringSeqProducer.initialDeclarations preserve=yes
    //##end WstringSeqProducer.initialDeclarations

    // Attributes
```

```
    // Relationships


    // Associations


    // Operations

    //## Operation: getWstringMsg
    //## Documentation:
    //      This operation is used to pull a CORBA Wstring
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
// Unsupported by ORBexpress at this time
//    void getWstringMsg(out PortTypes::WstringSequence msg, out CF::Properties
options);



    //##begin WstringSeqProducer.additionalDeclarations preserve=yes
    //##end WstringSeqProducer.additionalDeclarations

//  };

  //## AnyProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) an octet
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface AnyProducer : CF::Port {
    //##begin AnyProducer.initialDeclarations preserve=yes
    //##end AnyProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getMsg
    //## Documentation:
    //      This operation is used to pull a CORBA any
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
```

```
    //      source, destination, priority, etc.).
    void getMsg(out CF::DataType msg, out CF::Properties options);



    //##begin AnyProducer.additionalDeclarations preserve=yes
    //##end AnyProducer.additionalDeclarations

  };

  //## ShortSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a short
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface ShortSeqProducer : CF::Port {
    //##begin ShortSeqProducer.initialDeclarations preserve=yes
    //##end ShortSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getShortMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Shorts
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    void getShortMsg(out PortTypes::ShortSequence msg, out CF::Properties
options);



    //##begin ShortSeqProducer.additionalDeclarations preserve=yes
    //##end ShortSeqProducer.additionalDeclarations

  };

  //## BooleanSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a boolean
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface BooleanSeqProducer : CF::Port {
    //##begin BooleanSeqProducer.initialDeclarations preserve=yes
```

```
    //##end BooleanSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getBooleanMsg
    //## Documentation:
    //      This operation is used to pull a sequence of
    //      Booleans information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PullConsumer) object. The message
    //      being pulled has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
    void getBooleanMsg(out PortTypes::BooleanSequence msg, out CF::Properties
options);



    //##begin BooleanSeqProducer.additionalDeclarations preserve=yes
    //##end BooleanSeqProducer.additionalDeclarations

  };

  //## CharSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a
  //      character sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface CharSeqProducer : CF::Port {
    //##begin CharSeqProducer.initialDeclarations preserve=yes
    //##end CharSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getCharMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Chars
    //      information to be received or transmitted through
```

```
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    void getCharMsg(out PortTypes::CharSequence msg, out CF::Properties
options);



    //##begin CharSeqProducer.additionalDeclarations preserve=yes
    //##end CharSeqProducer.additionalDeclarations

  };

  //## LongLongSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a long
  //      long sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

//  interface LongLongSeqProducer : CF::Port {
    //##begin LongLongSeqProducer.initialDeclarations preserve=yes
    //##end LongLongSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getLongLongMsg
    //## Documentation:
    //      This operation is used to pull a sequence of Long
    //      Longs information to be received or transmitted
    //      through the RADIO from one object to the next
    //      "destination" (PullConsumer) object. The message
    //      being pulled has data and control information
    //      (classification, source, destination, priority,
    //      etc.).
// Unsupported by ORBexpress at this time
//    void getLongLongMsg(out PortTypes::LongLongSequence msg, out
CF::Properties options);



    //##begin LongLongSeqProducer.additionalDeclarations preserve=yes
    //##end LongLongSeqProducer.additionalDeclarations

//  };

  //## UlongSeqProducer Documentation:
  //      This interface is implemented by pull producers and
```

```
  //      used by a pull consumer that gets (pull) an
  //      unsigned long sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface UlongSeqProducer : CF::Port {
    //##begin UlongSeqProducer.initialDeclarations preserve=yes
    //##end UlongSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getUlongMsg
    //## Documentation:
    //      This operation is used to pull a sequence of
    //      Unsigned Longs information to be received or
    //      transmitted through the RADIO from one object to
    //      the next "destination" (PullConsumer) object. The
    //      message being pulled has data and control
    //      information (classification, source, destination,
    //      priority, etc.).
    void getUlongMsg(out PortTypes::UlongSequence msg, out CF::Properties
options);



    //##begin UlongSeqProducer.additionalDeclarations preserve=yes
    //##end UlongSeqProducer.additionalDeclarations

  };

  //## UlongLongSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) an
  //      unsigned long long sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

//  interface UlongLongSeqProducer : CF::Port {
    //##begin UlongLongSeqProducer.initialDeclarations preserve=yes
    //##end UlongLongSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations
```

```
    // Operations

    //## Operation: getULongLongMsg
    //## Documentation:
    //      This operation is used to pull a sequence of
    //      Unsigned Long Longs information to be received or
    //      transmitted through the RADIO from one object to
    //      the next "destination" (PullConsumer) object. The
    //      message being pulled has data and control
    //      information (classification, source, destination,
    //      priority, etc.).
// Unsupported by ORBexpress at this time
//    void getULongLongMsg(out CF_PortTypes::UlongLongSequence msg, out
CF::Properties options);



    //##begin UlongLongSeqProducer.additionalDeclarations preserve=yes
    //##end UlongLongSeqProducer.additionalDeclarations

//  };

  //## UshortSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) an
  //      unsigned short sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface UshortSeqProducer : CF::Port {
    //##begin UshortSeqProducer.initialDeclarations preserve=yes
    //##end UshortSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getUshortMsg
    //## Documentation:
    //      This operation is used to pull a sequence of
    //      Unsigned Shorts information to be received or
    //      transmitted through the RADIO from one object to
    //      the next "destination" (PullConsumer) object. The
    //      message being pulled has data and control
    //      information (classification, source, destination,
    //      priority, etc.).
    void getUshortMsg(out PortTypes::UshortSequence msg, out CF::Properties
options);
```

```
    //##begin UshortSeqProducer.additionalDeclarations preserve=yes
    //##end UshortSeqProducer.additionalDeclarations

  };

  //## StringSeqProducer Documentation:
  //      This interface is implemented by pull producers and
  //      used by a pull consumer that gets (pull) a stringt
  //      sequence from a pull producer.
  //## Category: Pull_Port_Producer_IDL_Components

  interface StringSeqProducer : CF::Port {
    //##begin StringSeqProducer.initialDeclarations preserve=yes
    //##end StringSeqProducer.initialDeclarations

    // Attributes


    // Relationships


    // Associations


    // Operations

    //## Operation: getStringMsg
    //## Documentation:
    //      This operation is used to pull a CORBA string
    //      information to be received or transmitted through
    //      the RADIO from one object to the next "destination"
    //      (PullConsumer) object. The message being pulled has
    //      data and control information (classification,
    //      source, destination, priority, etc.).
    void getStringMsg(out CF::StringSequence msg, out CF::Properties options);



    //##begin StringSeqProducer.additionalDeclarations preserve=yes
    //##end StringSeqProducer.additionalDeclarations

  };

};

#endif
```